

السلام عليكم ورحمة الله وبركاته أعزائي البصمجيين في كل بيت أعزائي البصمجيين في كل مكان
أهلا وسهلا بكم في بصمجاية جديدة في ليالي الفايغال الجميل وربنا يارب ما يقطع لنا عادة ولا يجرمني منكم ولا يحرملك مني ويارب
قدرنا دايماء علي فعل الخير وإبعدنا يارب عن أذية الغير وكتر يارب من اللايك والشير وإكفيننا يارب شر الظالم واللي مش فير ويارب
كل فايغال نكون معاكم أون اير.
ركز معايا يا حبيب أخوك ركزي معايا ياست الكل وبيننا نبداً صباح ليالي الفايغال

Lecture 1

What is the Data Structure?

Why we study Data Structure?

#what_is_the_program

البرنامج هو عبارة عن مجموعة أوامر نقوم بكتابتها للقيام بعملية معينة

- داخل البرنامج بيكون فيه مجموعة معينة من الداتا سواء موجودة في نفس البرنامج او موجودة في ملف خارجي
- فالتالي أي برنامج عندي عبارة عن : `code + data`
- لما باجي أعمل `run` للبرنامج بتاعي "ال `CPU` بيقوم بالعملية دي" بس فيه مشكلة وهي أن البرنامج بيبقا موجود على ال `Hard Disk` وغالبا بيبقا بطيء ف بيبطئ عملية ال `compile` للبرنامج
- عشان أحل المشكلة اللي فانت ف ال `CPU` بيبداً يتعامل مع ال `RAM`, ف أنا عندي الرام متقسمة ل 4 حاجات

1. Code
2. Global Variables
3. Stack
4. Heap

- ال `loader` بيقوم واخد الكود بتاعي ويرميه في جزئه المخصص في ال `RAM` وبيبدأ ياخد ال `data` ويحطها في مكانها المخصص في ال `RAM`

Data Structure

A data structure is a scheme for organizing data in the memory of a computer for efficient storage and efficient manipulation.

عندي two concepts مهمين ببص عليهم:

1. time complexity

- عبارة عن الوقت اللي يستغرقه في تنفيذ الكود بتاعي

2. memory complexity

- عبارة عن المساحة اللي هستخدمها للكود والدادنا بتوعي

- لما باجي اعالج مشكلة او اعمل برنامج يحل ليا حاجة معينة فيه بعض الخطوات اللي بمشي عليها

1. understanding the problem

- أول حاجة لازم افهم ايه المشكلة عشان اقدر اجيبها حل

2. designing a basic solution

- بعمل حل مبدئي للمشكلة بتاعتي

3. analysing the solution

- أحلل الحل بتاعي من حيث مميزات وعيوب والتايم والميموري

- بعد الخطوة دي عندي طريقين:

- بشوف لو ممكن ألاقي حل أفضل من الحل الأول بشوف الحل الجديد وأحلله

- بفضل أكرر الخطوة دي لحد ما ألاقي نفسي مش لاقى حل أفضل من اللي معايا حالياً

- بمجرد ما ألاقي حل مناسب للمشكلة بتاعي بأنسب تايم وميموري ومميزات وأقل عيوب

- ببدأ أكتب pseudocode للبرنامج بتاعي بعدين أبدأ أترجمه للكود بلغة البرمجة بتاعتي

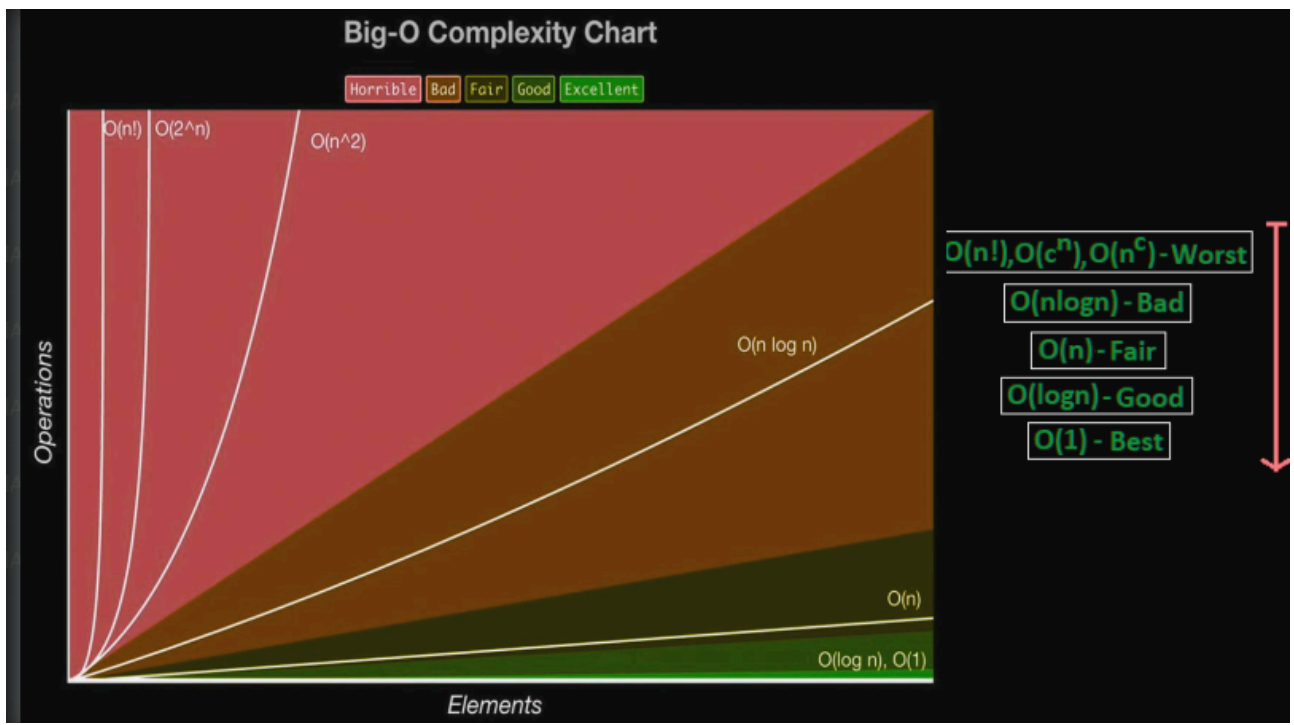
- الخطوة الأخيرة عبارة عن اني اتبست الكود بتاعي واشوف هل بينفذ اللي مطلوب منه زي ما أنا عاوز ولا فيه مشاكل

- عندي 3 مصطلحات بنستخدمهم عشان نشوف ال time complexity بتاعنا هيكون قد ايه

- أفضل حالة (Θ) Theta

- المتوسط (Ω) Omega

- أسوأ حالة (O) Big O (و دي اللي بنهتفم بيها لأن أنا ببقا محتاج أعرف أسوأ حاجة ممكن تحصل في البرنامج بتاعي ايه هي)



• ال $O(1)$

- مثلاً بنفذ عملية جمع أو طرح أو ضرب الخ.....
- بشوف كوندیشن هل ده مثلاً أكبر ولا ده أصغر الخ.....
- باختصار: أي حاجة بعيد عن اللوب

• ال $O(\log N)$

- لما يكون جوا اللوب وبزيد بمقدار ضرب في رقم معين أو القسمة على رقم معين
- بحيث أن اللوب بتاعتي مبتكونش ماشية linear
- مثال: الجورزم الباینري سيرش كل مرة بنقل ال search space للنص

```
// 1. Binary Search // o(log(n)) time - o(1) memory
if(is_sorted()) {
    int st = 0, nd = size - 1, mid;
    while(st < nd) {
        mid = st + (nd - st) / 2;
        if(arr[mid] == val)
            return mid;
        else if(arr[mid] > val)
            nd = mid - 1;
        else
            st = mid + 1;
    }
}
```

- افكر: لازم بيقا فيه صفة ال monotonic عندي عشان اقدر اعمل باينري سيرش

• ال $O(N)$

- لما بيكون عندي لوب وماشية بجمع أو طرح
- مثال: ال linear search أنا هنا بعدي على عنصر عنصر بشوف هل الرقم اللي بسيرش عليه موجود ولا لا

```
// 2. Linear Search // o(n) time - o(1) memory
for(int i = 0 ; i < size ; ++i)
    if(arr[i] == val)
        return i; // founded

return -1; // does not exist
}
```

- ف أحنا هنا لو خدنا بالنا من حاجة كانت موجودة فوق
- هل أنا لو عندي الاراي بتاعتي sorted, الأفضل استخدم linear search ولا binary search ؟!
- ف بالعقل كدا هيقا الاحسن اني استغل انها sorted واسيرش فيها ب $\log N$ بس
- و دي طريقة من طرق اني اختار افضل حل للمشكلة بتاعتي
- ال $O(N^2)$
- هنا بقا أنا بيكون عندي اتنين لوب جوا بعض كل واحدة فيهم بتمشي لحد n ف أنا هيقا عندي $n^2 = n * n$
- مثال على كدا ال bubble sort

```
// bubble sort is an efficient algorithm with memory o(1) but it's time is o(n^2)
// bubble sort is inefficient for large datasets.
auto bubble_sort = [&]() {
    for(int i = 0 ; i < (size - 1) ; ++i) {
        for(int j = 0 ; j < (size - i - 1) ; ++j) {
            if(arr[j] > arr[j + 1]) {
                T temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
};
```

- خلي بالك
- مش أي اتنين لوب جوا بعض يبقا $O(N^2)$
- ممكن يخدعك ويجيبلك اتنين لوب جوا بعض وتقوم رازعها N^2 مع أنها مش كدا خالص
- لو تفتكر ده مثلاً

```
for (int i = 0; i < n; i++) {
    while (j < n && nums[i] + nums[j] <= 10)
        j++;

    if (nums[i] + nums[j] == 10) {
        std::cout << "Pair found: " << nums[i] << " + " << nums[j] << " = 10\n";
        return true;
    }
}
```

Example 15

```
for(int i = 1; i ≤ n; i++)  
  for(int j = 1; j ≤ i; j++)  
    print "hello";
```

$T(N) = O(??)$

- في المثال ده هيبقا عندي التايم قد ابيه؟!
فكر سيكا كدا

Answer: $O(N^2)$

ال worst case عندي لما هتكون ال i ب n وال j هتيجي تلوب من 1 لحد n كدا هتكون $n * n$

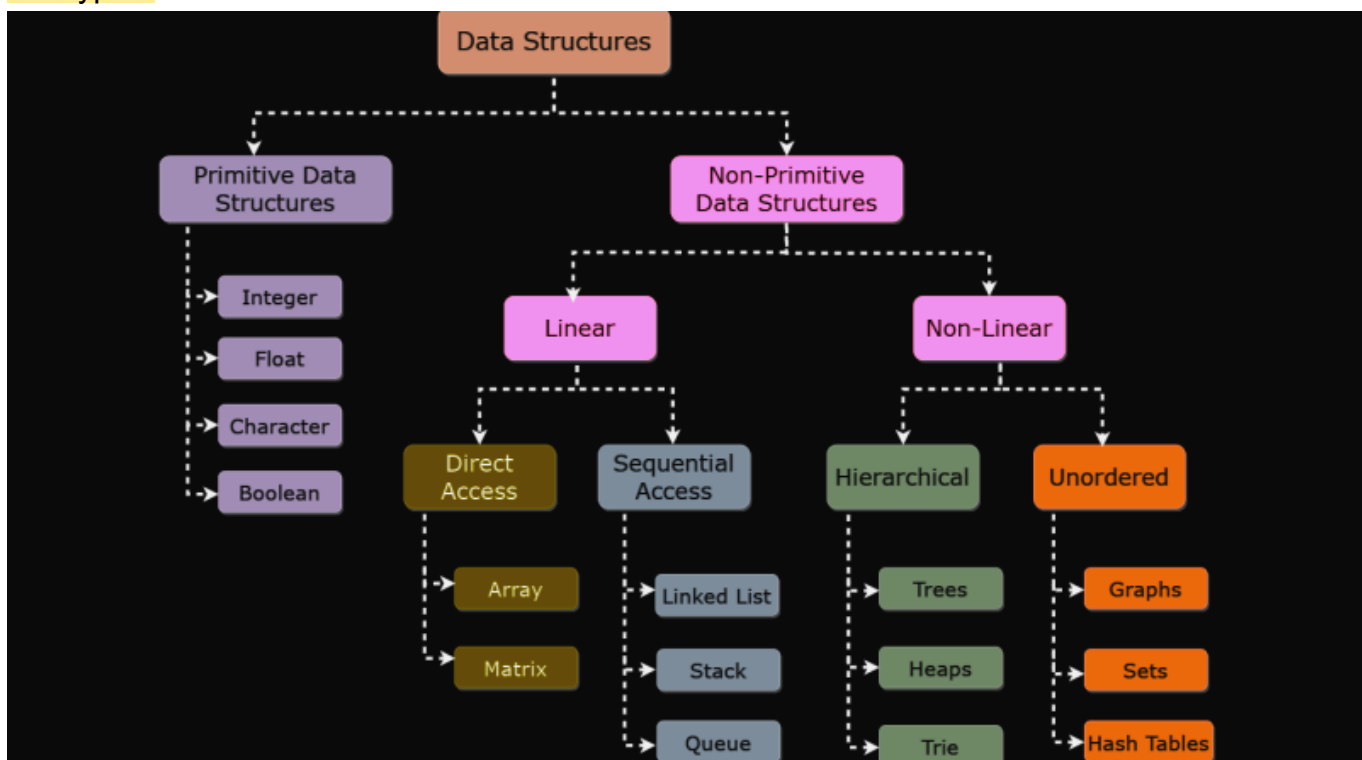
لو تفتكر فوق بقا السؤالين اللي سألناهم في أول البدء ده

- What is The Data Structure
- Why we study Data Structure
لو تفتكر برضو ف احنا جاوبنا عن الأولاني فوق
نيجي بقا نجاب عن الثاني

Why study Data Structure?

1. Efficiency
2. Problem-Solving Skills
3. Career Advancement

DS Types:



Linear Data Structure

- The data items are arranged in sequential order.

Non-Linear

- The data items are arranged in non-sequential order (hierarchical manner);

Static Data Structure

- the size of the structure is fixed.

Dynamic Data Structure

- the size of the structure is not fixed and can be modified.

Homogeneous data structure

- All the elements in Homogeneous data structure are the same.

Non-Homogeneous data structure

- The elements in Non-Homogeneous data structure may or may not be the same.

Abstract Data Type

- is the way we look at a data structure, **focusing on what** it does and **ignoring how** it does its job.

- هي طريقة بتخليني أشوف الهيكل الخارجي وهو عبارة عن ايه دون الدخول في أي تفاصيل وازاي هو مبني من جوا

Data structure is the physical implementation of an ADT.

Lecture 2

Static Array

is a basic data structure used to store a **fixed-size** collection of elements of the **same type** arranged in **contiguous memory locations**.

Characteristics

- Fixed Size
- Homogenous
- Indexed

- Efficient

Types

- 1D
- 2D
- Multi Dimensional

[Array Code](#)

Lecture 3

Dynamic Array

- is quite similar to a regular array, but its **size is modifiable** during program runtime.

[Code](#)

Aspect	Stack-Based Array	Heap-Allocated Array
Memory Location	The array resides on the stack, used for function call management and local variables.	The array resides on the heap, allowing dynamic memory allocation.
Memory Management	Memory is automatically allocated and deallocated based on the function's scope.	Explicit allocation and deallocation required using functions like <code>new</code> .
Scope	Limited to the function's scope.	Extends beyond the allocation scope, making it accessible from other parts of the program.
Size	Requires a known size at compile-time.	Size can be determined at runtime, providing flexibility.
Access Speed	Generally faster due to stack nature.	Slightly slower due to dynamic allocation.

RAII

- (Resource Acquisition Is Initialization)
- is a design principle in C++ where resources like dynamic memory are acquired and released automatically through constructors and destructors.

Rules of Three

- if a class manages a resource like dynamic memory
 - Destructor: To free the memory.
 - Copy Constructor: To correctly copy the array when copying objects.
 - Copy Assignment Operator: To correctly handle assignments between objects.
-

Lecture 4

Linked List

- flexible data structure that consists of elements called nodes, each containing data and a pointer to the next node.

Node

- Contains:
 - Data: The actual value or information that needs to be stored.
 - Next: A pointer (or reference) to the next node in the linked list.

Head

- The first node in a linked list (entry point);

NULL

- The last node of a linked list, which points to null, indicating the end of the list.

Types of Linked List

1. Singly Linked List

- collection of nodes where each node contains data and a reference to the next node in the sequence. This structure allows for a simple, **one-directional traversal** from the head to the end of the list.

2. Doubly Linked List

- Doubly linked lists expand on singly linked lists by having each node contain two references: one pointing to the next node and one to the previous node. This allows nodes to be **traversed in both forward and backward directions**.

3. Circular Linked List

- A circular linked list is a type of data structure where each node points to the next node, and the **last node points back to the first, forming a circle.**

[Singly Linked List Code](#)

[Doubly Linked List Code](#)

[Array VS Linked List](#)

Lecture 5

Stack

- data structures is a linear collection of elements that follows a particular order==(Last in First out) ==in which the operations of addition and removal of elements are performed.

[Stack Code Linked List Based](#)

[Stack Code Array Based](#)

Lecture 6

Queue

- s is a linear collection of elements that operates under the **First In, First Out (FIFO)** principle.

Types

- Simple
 - A simple queue operates on the FIFO (First In, First Out) principle, where elements are **added to the rear and removed from the front.**
- Circular
 - A circular queue is a more efficient version of the simple queue. It uses a **circular structure** to utilize memory optimally.

- Priority
 - In a priority queue, elements are not necessarily dequeued in the order they were enqueued, each element is associated with a priority.
- Deque
 - A deque allows insertion and removal of elements from both the front and the rear of the queue. This flexibility makes it a handy tool for various applications where elements need to be processed from both ends.

[Queue Code Linked List Based](#)

[Circular Queue Code Array Based](#)

[Priority Queue Linked List Based](#)

Lecture 7

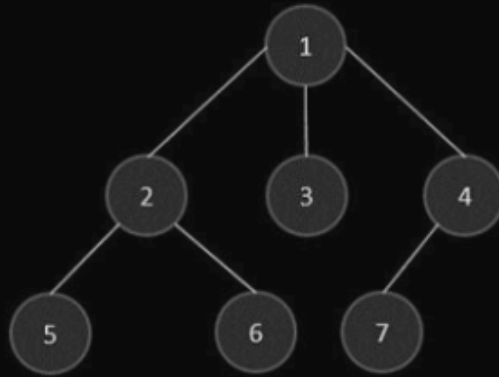
Trees

- General Trees
- Binary Trees
- BST
- AVL
- Heap
- Red-Black
- Segment Tree

General Trees

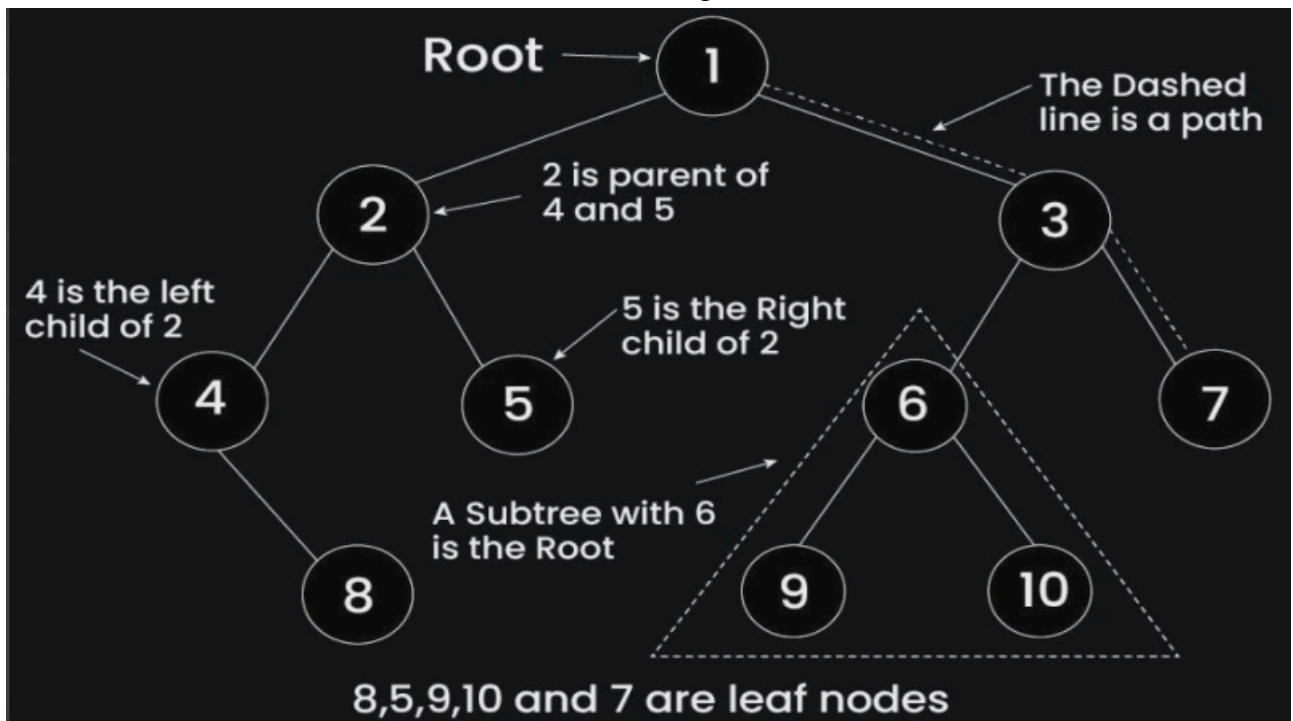
- tree data structures where the root node has minimum 0 or maximum 'n' subtrees.

The General trees have no constraint placed on their hierarchy.



Binary Tree

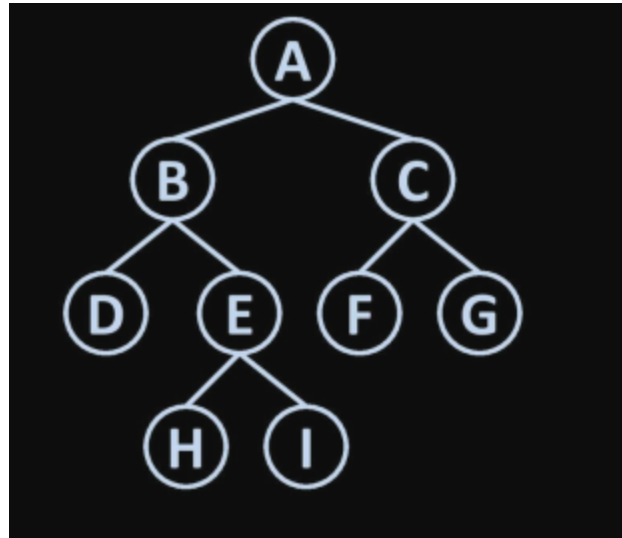
- a non-linear and hierarchical data structure (general tree), where each node has at most two children referred to as the left child and the right child



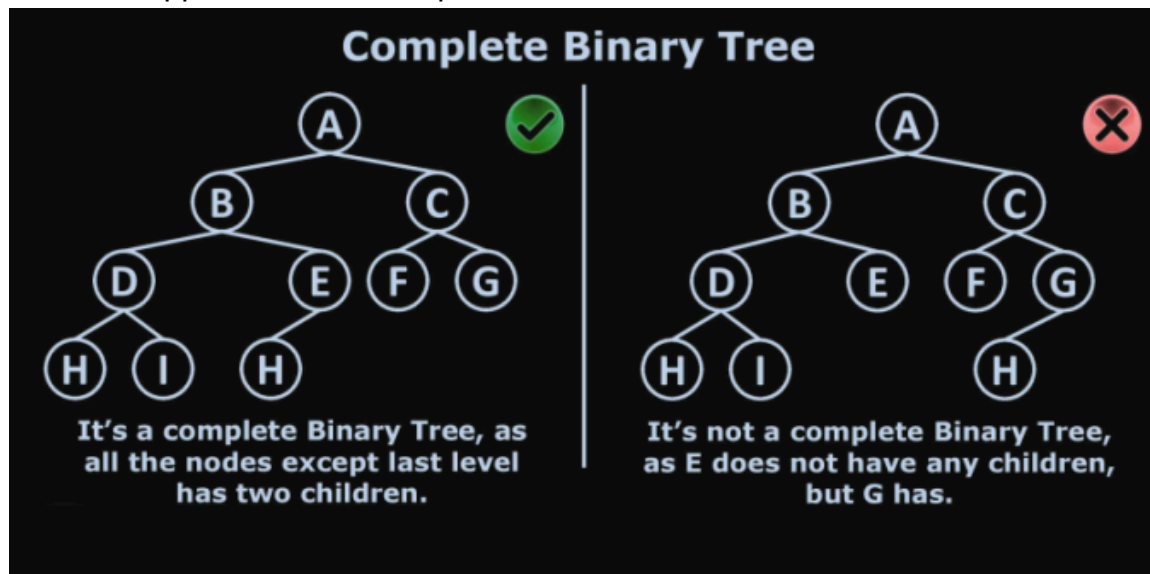
Variations

- Full Binary Tree
 - a binary tree type where every node has either 0 or 2 child nodes.

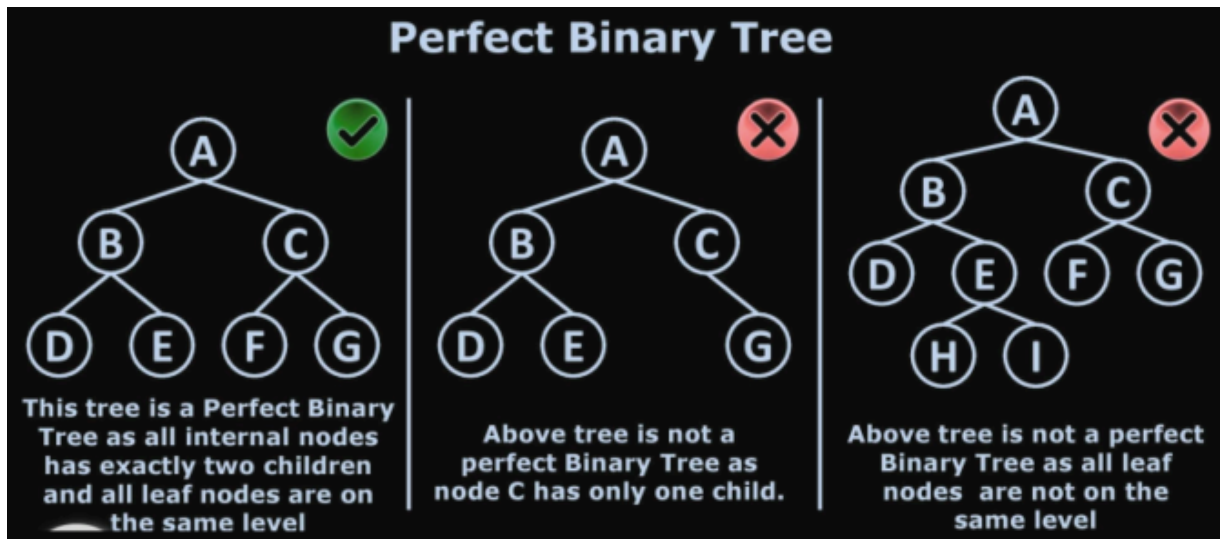
- هي تري بيكون كل نود عندها 2 نودز تحتها (زي ال root او ال internal node) أو معندهاش خالص (leaf)



- Complete Binary Tree
 - a binary tree that satisfies two properties.
 - every level, except possibly the last, is completely filled.
 - all nodes appear as far left as possible.



- Perfect Binary Tree
 - a binary tree type where all the leaf nodes are on the same level and every node except leaf nodes have 2 children



• يعني أنا كذا عندي ال complete بالبلدي كذا: بمشي من الشمال لليمين لو لقيت نود ملهاش نودز تحتها لازم بعد النود دي ميكونش فيه ولا نود تانية لها عندها نودز تحتها

• أما ال perfect لازم بيكا كله عنده 2 children nodes ماعدا ال leaf nodes
• وبكدا نقدر نقول

- All Perfect Trees are Complete Trees, but not vise versa

شوية مصطلحات لازم تبقا عارفها

- Nodes
 - The fundamental part of a binary tree, where each node contains data and link to two child nodes.
- Root
 - The topmost node in a tree is known as the root node. It has no parent.
- Parent Node
 - A node that has one or more child nodes.
- Child Node
 - A node that is a descendant of another node (parent)
- Leaf Nodes
 - A node that does not have any children or both children are null.
- Internal Node
 - A node that has at least one child. This includes all nodes except the root and the leaf nodes.
- Edge
 - A connection between two nodes.
- Path
 - A sequence of consecutive edges.
- Subtree

- The Subtree is a portion of the tree that includes a node and all its descendants.
- Level
 - Depth of the node from the root node (root node is at level 0)
- Degree
 - the number of children that a node has
 - indegree: عدد النودز اللي داخالي
 - outdegree: عدد النودز اللي رايحلها

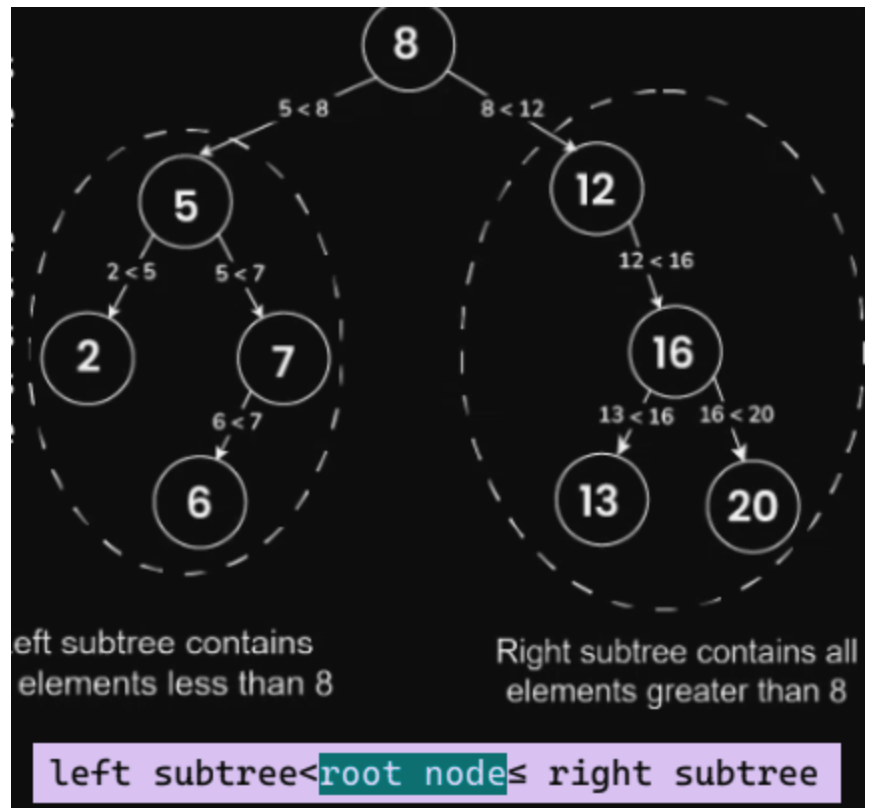
ركز بقا في الحتة دي عشان تبصمجها كويس

- preorder:
 - يعني pre يعني هطبع اللي انا واقف عليه الاول
- inorder:
 - يعني in يعني هطبع اللي انا واقف عليه في النص
- postorder:
 - يعني post يعني هطبع اللي انا واقف عليه في الآخر
 - وكدا كدا دايمًا ال left قبل ال right
 - تم البصمجة 

[Binary Tree Code](#)

Binary Search Tree

- هي باختصار سيكون عندي النود وكل اللي على الشمال اصغر من النود واللي على اليمين اكبر منها



- ميزتها انها بتخلي ال insert وال delete في $O(\log N)$
- بس فيها عيب قاتل وهو ان لو عندي القيم مثلا 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ال root كدا هلاقي كل النودز على يميني وعاملين linked list كدا طويلة عريضة مبالك بقا لو مليون نود بنفس الشكل كدا!!
- هنلاقي ال $\log n$ اتحولت n وال insert وال delete بقا كيبير جدا وساعتها بتبقا التري بتاعتي اسمها

Skewed

[BST Code](#)

The End

Remember

- 1% Chance
- 99% Faith

Good Luck <3